Carnegie Mellon University
Software Engineering Institute

Discovering DISCOVER

Scott R. Tilley

October 1997

TR

TECHNICAL REPORT
CMU/SEI-97-TR-012
ESC-TR-97-012

Technical Report
CMU/SEI-97-TR-012
ESC-TR-97-012
October 1997

Discovering DISCOVER

Scott R. Tilley

Reengineering Center
Product Line Systems

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

Table of Contents

List of Figures

Discovering DISCOVER

Abstract: This report describes investigations into DISCOVER, a modern software development and maintenance environment. The study is guided by a framework for classifying program understanding tools that is based on a description of the canonical activities that are characteristic of the reverse engineering process. Implications of this work for advanced practitioners, researchers and tool developers, and the framework itself are discussed.

1 Introduction

This report describes the use of a conceptual framework for the classification of reverse engineering tools and techniques (collectively known as support mechanisms) that aid program understanding [Tilley 96a]. The framework is based on a description of the canonical activities characteristic of the reverse engineering process. A descriptive model provides a means to classify different approaches to reverse engineering through a common frame of reference. It is important to note that the framework enables a comparison between different support mechanisms, not an evaluation of individual support mechanisms.

The framework was developed in three steps. The first step was an investigation into the underlying cognitive aspects of program understanding. The second step was the identification of the canonical activities that make up any reverse engineering process. These activities are data gathering, knowledge organization, and information exploration. The third step was a categorization of reverse engineering support mechanisms along several dimensions. Categories include domain applicability, task support, and toolkit extensibility.

This paper focuses primarily on the second and third parts of the framework for studying the capabilities of DISCOVER, a modern software development and maintenance environment. The first part of the framework, cognitive aspects, was not used extensively in the study because DISCOVER is primarily a tool for aiding program understanding, not a new comprehension model.

DISCOVER was chosen for this independent study for several reasons. It is one of the most advanced commercial environments currently on the market. Due to its pricing structure, it is unlikely that many researchers would have the opportunity to explore DISCOVER in an academic setting. It is a powerful environment that is representative of a new generation of tools that aid program understanding. As such, it provided an excellent opportunity to look at the state of the practice in program understanding tools. It also provided an opportunity to exercise and evolve the program understanding framework itself.

The next section provides a brief overview of DISCOVER. Section 3 describes the canonical reverse engineering activities and how DISCOVER provides support for them. Section 4 explores some of the quality attributes of DISCOVER, such as scalability and extensibility. Finally, Section 5 discusses the implications of this study for users, researchers and tool developers, and the program understanding framework itself.

2 DISCOVER

DISCOVER™ is a "Development Information System" from Software Emancipation for managing software applications [Software 96]. DISCOVER is a good example of a new generation of reverse engineering environments. Central to its use and philosophy is a repository, called the Information Model (IM), which contains complete dependency information about the entire subject system. Creation of the IM is guided by a description of the mappings between the logical and physical structure of the subject system. It is initially populated by parsing the source code (DISCOVER supports both C and C++) and is incrementally updated from then on.

There are five major application sets in DISCOVER that make use of the IM. Each application set is composed of a series of modules that address one or more development tasks. Of the five, DEVELOP/set is the one most geared towards program understanding. Parts of REENGINEER/set are also applicable to program understanding tasks. However, DEVELOP/set was the focus of this study.

2.1 DEVELOP/set

DEVELOP/set is a group of applications used for typical software development activities. It is the application set most targeted to supporting program understanding. DEVELOP/set consists of four modules: PROGRAM/sw, DESIGN/sw, DEBUG/sw, and DEBUG+/sw.

2.1.1 PROGRAM/sw

Of the four modules in DEVELOP/set, PROGRAM/sw is the module most directly aimed at aiding program understanding. It provides two tools, the Browser and the Viewer, which are the primary user interfaces of DISCOVER. The Browser serves as the navigation tool, while the Viewer serves as the main presentation tool. The Viewer is used to display source code (and associated artifacts) in a textual or graphical manner. PROGRAM/sw also provides tools for "what if" change and impact analysis, and for automatically propagating desired changes throughout the subject system. The Impact Analysis tool reports on the system-wide impact of any proposed change before the change is submitted. The Change Propagation tool checks out affected files from the existing configuration management (CM) system and implements the approved change. These tools are discussed in more detail in Section 3.3.

2.1.2 DESIGN/sw

The DESIGN/sw module expands the functionality of the Viewer by providing the ability to create and modify graphical views, such as subsystem diagrams, flow charts, and class inheritance diagrams. Enhanced source code editing is also provided through syntax-directed versions of the Emacs and vi editors. Modification of any of the diagrams supported by DESIGN/sw results in automatic modification (or incremental generation) of the associated source code to reflect the changes. DESIGN/sw also supports forward engineering by providing a facility for the creation of standard code fragments based on user-specified templates.

2.1.3 DEBUG/sw

DEBUG/sw is an integrated interface to industry-standard debuggers, such as gdb and dbx. Debugging commands are entered using a graphical or command-line interface, translated, and passed on to the underlying debugger. DEBUG/sw also provides a macro language for writing debugging scripts. The language supports multi-line macros that accept passed-through arguments, evaluate expressions, and use conditional statements. The macros are stored as files, so custom libraries of debugging procedures can be developed.

2.1.4 DEBUG+/sw

DEBUG+/sw enhances DEBUG/sw through support of mixed-mode debugging. The interpreter allows one to isolate the function to be interpreted and run it within a compiled executable. The resulting "on the fly" debugging of interpreted and compiled code eliminates the need to recompile and re-link, thus reducing cycle time. This functionality is provided using the industry-standard debuggers supported by DEBUG/sw and special libraries, supplied as part of the module, that are linked into the application program.

2.2 CM/set

CM/set provides configuration management support by integrating with standard CM systems such as RCS and SCCS. This approach means that existing CM systems can be leveraged, and users need not learn a new CM system. CM/set provides an interface to the underlying system similar to the manner in which DEBUG/sw provides an interface to underlying debugging systems.

2.3 REENGINEER/set

Although part of REENGINEER/set can be used for aiding program understanding, it is geared more towards altering the subject system through transformations. The REENGINEER/set applications restructure monolithic applications into separate modules, or into components that incorporate a subset of the functionality of the original software. REENGINEER/set consists of four modules: EXTRACT/sw, PACKAGE/sw, PARTITION/sw, and SIMPLIFY.H/sw.

The EXTRACT/sw module extracts logical subsystems from a subject system. The IM is used to analyze the system's source code and subdivide it into a series of subsystems that graphically depict the logical partitioning of the system's functionality. A "dead code" subsystem can be identified through dormant code analysis.

PACKAGE/sw subdivides a subject system into logical components based on the dynamic relationships of the functions and data in the program. This module can be used to generate functional subsets of an existing application, to identify and modularize library code, or to break an application into segments before reengineering it using a different paradigm.

PARTITION/sw is a module for restructuring a subject system. Source code can be physically rearranged into a new structure that more accurately reflects its logical structure, as identified using the EXTRACT/sw module. Such restructuring can often make maintenance easier and can facilitate program understanding by reducing the code clutter.

The SIMPLIFY.H/sw module is used to restructure a subject system's header files. It uses the IM to reduce to a minimum the number of header files included in source files. This results in fewer recompilations after changes are made to interfaces defined in header files.

2.4 DOC/set

DOC/set is used to maintain consistency between source code and documentation. It consists of DOC/sw.FRAME and REPORT/sw.

DOC/sw.FRAME is used to establish associations between source code and Adobe FrameMaker documents. The associations are represented as links that facilitate navigation between related elements. The PROGRAM/sw module uses these links when performing Impact Analysis and Change Propagation operations. After a link is established, it is treated in much the same way as a relationship between source code artifacts. The difference is that relationships between source code artifacts are discovered automatically during parsing and model building, while relationships between source code and documentation must be manually created.

The REPORT/sw module generates system-wide documentation that represents the system as it currently exists, reporting what it contains and how entities are related. Links are automatically created between source code and the generated documentation. These links are used in the same way as those in DOC/sw.FRAME.

2.5 ADMIN/set

ADMIN/set provides system administration facilities for setup, maintenance, and usage of DISCOVER. It currently consists of just the ADMIN/sw module. ADMIN/sw provides facilities for managing the Project Definition Files discussed in Section 3.2.

3 Canonical Activity Support

As discussed above, a key component of the program understanding framework is the set of canonical activities that are characteristic of any reverse engineering exercise. This section describes how DISCOVER provides support for these activities.

3.1 Data Gathering

To identify the artifacts and relationships of a system and use them to later construct and explore higher-level abstractions, raw data about the system must be gathered. Hence, data gathering is an essential reverse engineering activity. It is usually, but not always, the first step. The raw data is used to identify a system's artifacts and relationships; without it, higher-level abstractions cannot be constructed and explored. Techniques used for data gathering include static source code analysis (such as parsing), dynamic analysis (such as profiling), and informal extraction (such as interviewing).

DISCOVER provides data gathering capabilities through the use of the underlying Gnu compilation system from the Free Software Foundation. It supports both C and C++. Parsing source code is the primary means of populating the IM. The static data gathered is extensive and detailed: both inter- and intra-module dependencies are extracted. This aids coarse-grained program understanding at the structural level as well as fine-grained understanding at the algorithmic and data structure level.

DISCOVER also uses information produced by other tools, such as Pure Atria's Quantify, as data to support dynamic analysis and interpretive debugging in the DEBUG/sw and DEBUG+/sw modules.

Finally, data representing relationships not easily extractable from the source code, such as a semantic dependency between software and its associated documentation (or vice-versa), can be gathered from the user and represented as associations. This information is used in the DOC/set application (as discussed in Section 2.4).

3.2 Knowledge Organization

For successful program understanding, data must be in a form that facilitates efficient storage and retrieval, permits analysis of artifacts and relationships, and reflects the users' perceptions of the system's characteristics. This form is usually based on a data model. A data model enables one to understand the essential properties and relationships between artifacts in a system. Without a model, raw data is almost impossible to understand. Knowledge organization techniques are used to create, represent, and reason about data models.

DISCOVER relies on the IM to organize knowledge. The IM can be viewed as an associative, consolidated, persistent repository that keeps track of every artifact related to the software project, regardless of its size or apparent complexity, and every relation between artifacts. The IM can reside on either a designated or virtual server. It is essentially a distributed database that is used by all DISCOVER applications.

One or more Project Definition Files (PDF) guide the structure of the IM. A "project" in DISCOVER parlance is a user-defined collection of files. It can mimic the current physical directory structure, or it can impose a user-specified logical structure in its place. The PDF specifies the mappings between the logical names of project components and their actual physical locations in the underlying file system. A PDF can be somewhat complex to create, but it does provide a sophisticated mechanism for abstracting away from the structure of the file system (which can become quite complicated and "dirty" over time) to a more logical view of the application.

Two types of binary files physically represent the IM: (usually) one "pmod" file that contains information relevant to every file in a project, and one or more "pset" files that contain information relevant to a specific file in a project. The pmod file is stored at the top-level directory for the project. It contains project-wide entity-relation cross-reference information, which permits rapid access to coarse-grained relations. The pset files contain file-specific information and are (by default) stored in the same directory as the source code. For example, for a source code file foo.c, there would be a file called foo.c.pset containing complete parse details about the contents of foo.c. It will also contain cross-reference information about the file's relation to other files. In essence, pmod files are used to capture information about global entities in the subject system (such as functions or classes), while pset files are used to capture information about entities whose scope is not global (such as automatic variables).

Parsing the subject system's source code initially populates the IM, resulting in a complete and very detailed representation of the system's structure. After changes are made to the system, the IM can be regenerated from scratch or incrementally updated. DISCOVER is able to adjust the contents of the model to reflect modifications made during development. Incremental updating of the model works whether the code is modified using one of DISCOVER's toolsets or not. For a multi-user environment, DISCOVER separates the shared area (code visible to an entire team) and an individual's private work area.

3.3 Information Exploration

Information exploration is a composite activity of navigation, analysis, and presentation. Because the majority of program understanding takes place during information exploration, it is perhaps the most important of the three canonical reverse engineering activities. Data gathering is required to begin the reverse engineering process, and knowledge organization is needed to structure the data into a conceptual model of the application domain and subject system. But the key to increased comprehension is exploration that facilitates the iterative refinement of hypotheses. Exploration includes navigating through the web that represents the information related to the subject system, analyzing and filtering this information with respect to domain-specific criteria, and using presentation mechanisms to clarify the resultant information.

3.3.1 Navigation

Large software systems, like other complex systems, are non-linear and may be viewed as consisting of an interwoven and multidimensional web of information artifacts. The web's links establish relationships between the artifacts. These relationships can be component hierarchies, inheritances, data and control flow, and other relationships generated as part of the reverse engineering and software development processes. Navigation allows software engineers to traverse this "information web" as part of their exploratory understanding activities.

The information navigation activity can itself be subdivided into selection, editing, and traversal. Selection is one of the most important of all canonical activities. It is related to the essence of program understanding: identifying artifacts and understanding their relationships. This is essentially a pattern matching activity at various abstraction levels. Identifying artifacts involves finding artifacts of interest in the subject system or in the database used to represent the system. It requires a query mechanism that enables users to specify attribute patterns that are used to identify artifacts in the database that satisfy the search criteria. The type of pattern recognition provided can range from recognition of simple regular expressions, such as that provided by the Unix grep tool, to more advanced capabilities such as plan recognition.

Editing is an activity that can alter the knowledge organization structure, sometimes as a by-product of information navigation. For example, through editing activities a user may create user-specified subsystem constructs that are logical (but not physical) representations of the system. Traversal is the action of moving from one artifact to another in the information space. For example, following links that represent relations such as "calls" involves traversal.

Two tools from the PROGRAM/sw module provide the primary means of navigation in DISCOVER: the Browser (also known as the Project Browser) and the Viewer. The Browser operates in two modes: Browse and Scan. In Browse mode, information is displayed as it exists in the underlying physical directory structure; the PDF is used to filter out non-project files. In Scan mode, information is displayed as it exists in the IM. Figure 1 on page 12 illustrates the Browser operating in Scan mode.

The Browser is structured as a table with four resizable columns: Categories, Elements, Ask, and Results. In Scan mode, the Categories column represents different types of entities in the IM; it does not change. The Elements column is dynamically updated to contain instances of the selected Categories. The Ask column contains queries the user can launch concerning the selected Elements. This is the primary means of selecting artifacts in the IM. The queries are extensive but fixed. They permit the user to see where entities are defined, used, included, and so on. However, because they do not change, some queries do not make sense for certain entities. For example, asking for “Show Friends” to display the friend functions of a C++ method does not make sense if the element selected is not a C++ class.

The Results column contains entities from the IM that satisfy the query. They may be clustered into logical entities called “Groups,” which can be used as first-class entities in subsequent queries or analyses. In this manner the user can edit the IM to construct logical groupings of entities that are not necessarily reflected in the physical structure of the subject system. Pattern matching filters are available to refine the queries. A Query tool is also available for context-sensitive and attribute-based searches. For example, the user can search for all files with a specified creation date. At any time the user can select one or more entities displayed in the Browser and graphically view their relationships to other entities using the Viewer. Users can traverse the implicit links between the entities in the Results column and the associated source code if the entity represents a file.

As discussed below, the Viewer is primarily used for information presentation. However, it also supports information navigation through its tight integration with the Browser. While the Browser is used to navigate through the entire project, the Viewer can be used to navigate in a single source file. The Viewer comes equipped with enhanced versions of the Emacs and vi editors that provide hyperlinks between source code files and associated documentation. The Decorated Source option highlights syntactic entities in the viewer, such as macros and keywords. It also provides a mini-Browser of its own for rapid navigation.

3.3.2 Analysis

The critical step that derives abstractions from the raw data is analysis. Software engineers use the resultant information to further understand the system. There are many forms of analysis possible, such as computing metrics or slicing. The type of analysis supported is closely related to the abstraction level provided by the pattern recognition capabilities of the tool. Program understanding techniques can consider source code in increasingly abstract forms: raw text, preprocessed text, lexical tokens, syntax trees, control and data flow graphs, program plans, architectural descriptions, and conceptual models. The more abstract forms entail additional syntactic and semantic analysis that corresponds more to the meaning and behavior of the code and less to its form and structure. Different levels of analysis are necessary for different users and different reverse engineering activities.

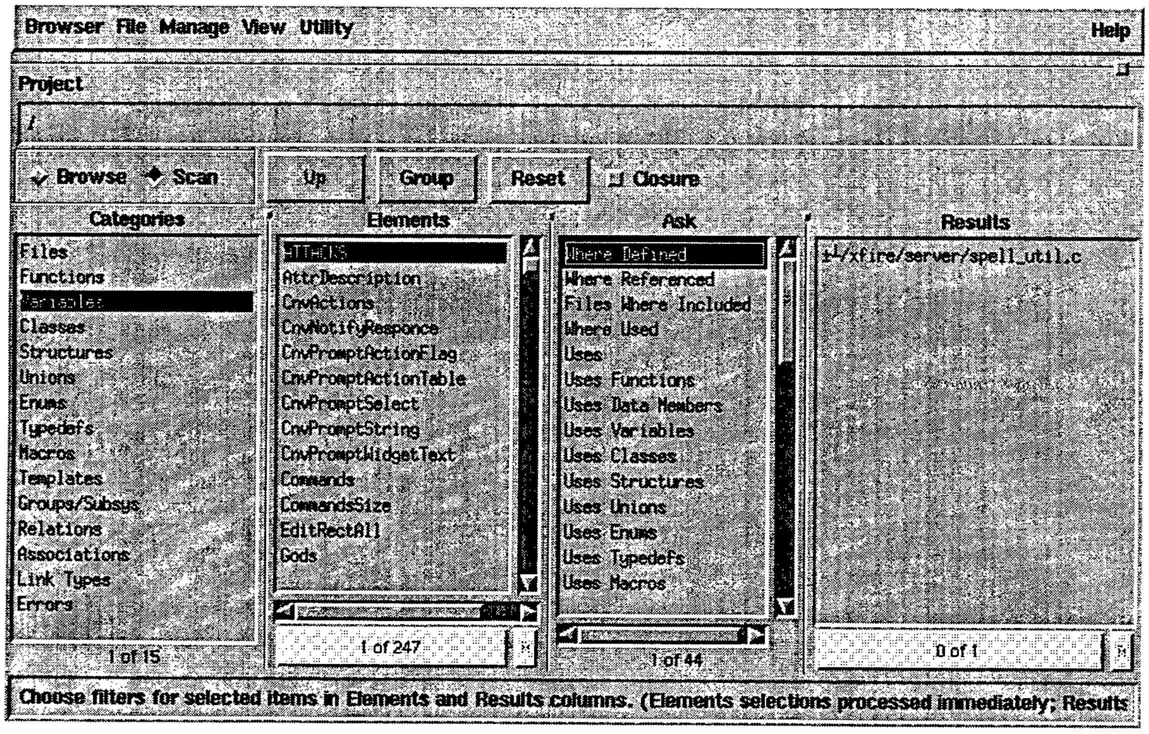
Many of DISCOVER's information analysis capabilities are provided by PROGRAM/sw's Impact Analysis and Change Propagation tools, and by the DEBUG/sw and DEBUG+/sw modules. As mentioned in Section 2.1, the debugging tools provide a common interface to the underlying debugger. Mixed-mode execution of interpreted and compiled code enables considerable runtime analysis to be performed quickly and easily.

Impact Analysis analyzes the impact of a proposed change throughout the entire project. This type of analysis can significantly reduce the number of errors introduced by incomplete or erroneous changes. If the scope of the changes is acceptable, the Change Propagation tool can be used to automatically implement the change throughout the entire system. The types of changes that can be automated in this fashion include changing the name of a variable and adding new parameters to a function. Affected files are checked out of the underlying CM system as needed.

The change management process relies on two types of relations: Hard Associations and Soft Associations. Hard Associations are those that are automatically gathered by DISCOVER from the source code and maintained in the IM. Soft Associations are those that are manually entered by the user, for example using DOC/set, and from there after are tracked in the IM. Soft Associations are used to maintain semantic dependencies between entities such as source code and related documentation.

The Group entity makes it possible to quickly carry out other common analysis operations, such as locating all functions referenced but not defined in the project. This is done by first creating a Group representing all functions defined in the project, as selected in a few mouse clicks in the Browser. A second Group is created that represents all functions referenced in the project. A set difference operation is then used to identify the resulting collection of undefined functions. Such functions are often library routines and may be of concern when porting the subject system.

The pattern matching capabilities of DISCOVER are primarily geared towards program analysis, where the abstraction level is at the syntactic programming language level and source-code tokens, such as keywords and identifiers, are the artifacts manipulated.



Choose filters for selected items in Elements and Results columns. (Elements selections processed immediately; Results

Figure 1: DISCOVER Browser

3.3.3 Presentation

Because people often use visual metaphors for communicating and understanding information, it is important to use flexible presentation mechanisms. Most reverse-engineering systems, including DISCOVER, provide the user with fixed presentation options, such as cross-reference graphs or module-structure charts. Even though the producers of the system might consider fixed options to be adequate, there are always users who want something else. Ideally, it should be possible to create multiple, perhaps orthogonal, structures and to view them using a variety of mechanisms (for example, different graph layouts provided by external toolkits).

In DISCOVER, information presentation is accomplished primarily using the Viewer. It provides both textual and graphical representations of the subject system. Information presented to the user is always up to date: The views are generated dynamically from the IM. Textual views are provided using an enhanced version of the Emacs and vi editors. The Viewer can display source code files in Outline mode and use elusion to hide details as desired.

A considerable number of graphical views of entities and relations are available. Subsystems, which are user-defined collections of entities, are treated as a special type of Group. Subsystem diagrams show the relationship between these entities. Similar entity-relation diagrams are available for other types of entities in the IM, such as classes. For example, class inheritance diagrams provide a view of the inheritance hierarchy. For finer grained information, flow charts can be created to display the logic and control flow of selected functions.

The diagrams are color-coded to indicate different types of relations. Icons are used to represent the different types of entities in the IM. Visual cues are provided to indicate when the current diagram is incomplete and that more information is available. This is an important feature, as it ensures that the diagrams will not mislead the users into making decisions based on what they perceive as complete information.

4 Quality Attributes

Part of the program understanding framework is a descriptive model that characterizes a reverse engineering support mechanism based on a hierarchy of attributes. A *quality attribute* is defined by Barbacci, et al., as a system requirement that is essentially non-functional in nature [Barbacci 95]. Examples of quality attributes include dependability, extensibility, and usability. This section briefly describes selected quality attributes of DISCOVER that were found to be of particular importance.

4.1 Applicability

This quality attribute is related to the domain in which the tool is applicable. In this case, the word "domain" is over-burdened. It can refer to application domain, implementation domain, and issues of scale.

DISCOVER currently supports C and C++. There is no implicit limitation on the type of application for which DISCOVER is appropriate. The IM is two-tiered (global "pmod" files and local "pset" files) and distributed, which helps make it scalable. It has been used on systems with millions of lines of C/C++ code. In many ways, DISCOVER is best suited to large projects. The benefits that accrue from using a large-scale centralized repository really only make themselves felt when the size of the subject system goes beyond that which a small team can keep track of manually.

4.2 Extensibility

DISCOVER is essentially a closed system. The IM is proprietary and no published API is available to the average user, although point integration is possible (such as that provided with FrameMaker). There is no scripting interface that would enable end-user programming of its modules.

The suite of tools DISCOVER provides is extensive, but fixed. The queries available in the Browser cover most of the likely questions a C or C++ programmer would ask, but there is no way to extend this set.

DISCOVER supports varying degrees of automation. For example, Groups can be constructed manually, subsystems can be extracted semi-automatically (guided by parameters set by the user), and header files can be simplified automatically.

4.3 Miscellaneous

DISCOVER currently runs on the Solaris and HP-UX operating systems. It takes approximately 85M of disk space for a typical installation. To process upwards of 300K

LOC, 64M of main memory and 140M of both temp and swap space are recommended. There are no external requirements for third-party software. DISCOVER is written in C++ and is developed and maintained using itself; this is an important testament to its usefulness.

5 Discussion

This look at the capabilities of DISCOVER, as viewed through the lens of the program understanding framework, has raised many issues about reverse engineering environments that support program understanding. This section discusses some of the implications of this work on users of such support mechanisms, on the program understanding research community and tool developers, and on the evolution of the program understanding framework itself.

5.1 Implications for Users

The report *Coming Attractions in Program Understanding* identifies emerging technologies in program understanding that may have significant impact on advanced practitioners in the next five years [Tilley 96b]. Of the three areas discussed in that report, the section on development of support mechanisms is the most relevant for this discussion of DISCOVER's capabilities. In particular, DISCOVER currently provides at least partial support for at least five of the emerging technologies, only two of which were expected to appear in 1997. They are

1. Leveraging mature technology: Existing compilers are tailored for use in data gathering.
2. Data filtering: Several techniques for filtering information are provided.
3. Advanced modeling techniques: Groups and the IM provide a logical view of the subject system.
4. Scalable knowledge bases: The IM can handle upwards of one million lines of code.
5. Automation levels: The user can select manual, semi-automatic, or automatic operations in some instances.

It is clear that many of DISCOVER's advances are in the area of knowledge organization, which is not surprising given its emphasis on the IM.

The implication for advanced practitioners is that advanced commercial environments like DISCOVER provide a production-quality amalgam of several recent research efforts into reverse engineering environments. DISCOVER is scalable (able to process very large source files), robust (able to process industrial code), and relatively easy to use. The notable exception to the latter quality attribute (usability) is in the initial creation of the IM. This step requires considerable domain knowledge and intimate familiarity with the system environment in which DISCOVER is deployed. Model creation has proven to be somewhat challenging in a few instances where DISCOVER was introduced into a real-world environment.

There are a few areas where DISCOVER does not yet provide some of the desired capabilities outlined in the report. Perhaps the most important is end user programmability. As discussed above, there is currently no way for a user to write scripts to tie some of the DISCOVER tools together. This limits the capability of DISCOVER by forcing the user to redo tasks, such as creation of and operation on Groups, that could be automated in some instances. The other important limitation related to this is the closed nature of the IM; third-party tools cannot yet be integrated into the DISCOVER toolset by the end user. This limits the capability of DISCOVER by not allowing users to run tools, such as those that calculate site-specific metrics, using applications they may already have in-house.

5.2 Implications for Researchers and Tool Developers

The emergence of commercial tools such as DISCOVER has implications for researchers and tool developers working in program understanding. Research tools should "push the envelope" in some way, advancing the state of the art beyond the current state of the practice. This means that any new tools proposed by the research community must necessarily provide capabilities not found in commercial offerings. Since DISCOVER provides such a rich feature set, it has raised the bar of what should be considered *de facto* tools in a reverse engineering environment.

However, as discussed in the previous section, there are still many avenues of exploration that would benefit from more attention by researchers. One of the most promising areas is information navigation and presentation. When DISCOVER is used on very large systems, the issue of scale becomes very important. Visualization techniques are needed to sort and filter information.

Web-based interfaces would go a long way towards making the capabilities of tools like DISCOVER available to a wider group of users. The application domain in which developers work is changing rapidly, from traditional monolithic programs to client-server to three-tiered net-centric applications that rely on distributed object technology. A tool such as DISCOVER can be used on this type of application, but much more work is needed in this area.

There is another issue that arose from the study of DISCOVER that is applicable to all researchers: deployment. The issue of successfully deploying a commercial tool in an industrial setting is not new. Researchers in the CASE tool community have been struggling with the adoption issue for a long time. New technology is only successful if it easily integrates with existing tools and processes. Forcing users to adopt radically different ways of doing their jobs rarely succeeds. It is unfortunate that many developers will only give a tool a short window of opportunity to succeed. If they cannot get the tool up and running in ten minutes and see real results, without looking at the manual, they will often abandon the tool. For tools as complex as DISCOVER this is a very real concern. It means the potential benefits of using the tool may not be realized. Perhaps one way of addressing this problem is by promoting the capabilities of such support mechanisms as services rather than tools. Another way may be to incorporate more automated support for initial setup and integration with the target environment. Automatic "wizards" such as those found in popular PC office software might be used to provide more hands-on guidance when domain experts are not available. The crux of the matter, from a researcher's point of view, is that no matter how powerful the tool or technique advocated, it will not be successful until it is merged into the normal activities of the users.

5.3 Implications for the Program Understanding Framework

The underlying cognitive aspects of program understanding remain fundamental to research into reverse engineering. However, for comparing individual support mechanisms, this part of the framework was de-emphasized in this study. There is, however, an opportunity for further investigation into the link between cognitive models and the types of processes a reverse engineering tool supports, or even mandates. Storey reports about some initial work in this area [Storey 97].

In evaluating the utility of the program understanding framework for classifying reverse engineering tools, several questions must be addressed. Did the investigation into the tool's capabilities reveal any fundamental flaws in the framework? Is it complete? That is, are there features offered by the tool that could not be mapped into the canonical activities and/or quality attributes sections of the framework? Conversely, are there areas of support described in the framework that do not seem readily available in the tool in question? Finally, how easy was it to use the framework to classify the capabilities of the tool?

In this instance, DISCOVER proved to be a very useful case study. There were not really any capabilities that DISCOVER offered that could not be fit into the program understanding framework—at least none of DISCOVER's modules that are geared towards program understanding. DISCOVER provides a richer set of features than just program understanding aids, such as tools for reengineering and document management, that were outside the scope of this study because they are not covered by the program understanding framework. Most of the categories in the framework were applicable to DISCOVER. The mapping of tool features to canonical activities and quality attributes was relatively easy to do. Nonetheless, based on this study of DISCOVER (and of other tools), the structure of the program understanding framework has changed.

The section on quality attributes evolved from what was originally a descriptive model of the characteristics of reverse engineering environments. This change made the description of some of the capabilities investigated, such as information analysis, easier to provide under the canonical activities rubric. There is still a need for a scenario-driven version of the framework that is more geared towards typical user maintenance tasks. For example, a high-level scenario of "porting" could be refined into maintenance tasks, such as "identify external library dependencies." Each such task is composed of one or more of the canonical activities described in Section 3. This enhancement of the framework would make it more amenable to users, as opposed to researchers.

Some of the canonical activities and quality attributes will be explored in more detail for the next study. For example, the analysis of DISCOVER's information presentation system gave some impetus to taking this part of the framework down a few levels of detail. This would aid the discussion of the importance of visual cues and different types of diagram techniques. Other quality attributes, such as usability and ease of deployment, are also candidates for further examination.

More investigations of other popular reverse engineering systems, guided by the evolving program understanding framework, are currently underway. This includes both research prototypes and commercial offerings. In addition, two "baseline" environments are being investigated: the traditional Unix environment and its generic toolset, and a newer type of environment exemplified by tools such as Microsoft's Visual Studio [Microsoft 97]. The latter will be used as the basis for an investigation into leveraging HTML and the Web as the infrastructure for a new class of reengineering tools [Tilley 97].

5.4 Acknowledgements

Software Emancipation was kind enough to provide an evaluation copy of DISCOVER and initial training in its use. Without their help, this work could not have taken place. Discussions with Brian Gill-Price of proServices were invaluable in learning about experiences deploying DISCOVER in real-world settings. Ed Morris and Dennis Smith of the Software Engineering Institute provided valuable comments on early drafts of this paper.

References

[Barbacci 95]	Barbacci, M.; Klein, M. H.; Longstaff, T. H.; & Weinstock, C. B. *Quality Attributes* (CMU/SEI-95-TR-021). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1995.
[Microsoft 97]	Microsoft. *Visual Studio* [online]. Available WWW <URL: http://www.microsoft.com/vstudio/> (1997).
[Software 96]	Software Emancipation. *The DISCOVER Development Information System* (Version 4.0) [online]. Available WWW <URL: http://www.setech.com> (1996).
[Storey 97]	Storey, M. A.; Wong, K.; & Müller, H.A. "How Do Program Understanding Tools Affect How Programmers Understand Programs?" pages 12-21. *Proceedings of the 4th Working Conference on Reverse Engineering (WCRE '97).* Amsterdam, The Netherlands, October 6-8, 1997. Los Alamitos, CA: IEEE Computer Society Press, 1997.
[Tilley 96a]	Tilley, S. R.; Paul, S.; & Smith, D. B. "Towards a Framework for Program Understanding," pages 19-28. *Proceedings of the 4th Workshop on Program Comprehension (WPC '96).* Berlin, Germany, March 29-31, 1996. Los Alamitos, CA: IEEE Computer Society Press, 1996.
[Tilley 96b]	Tilley, S. R. & Smith, D. B. *Coming Attractions in Program Understanding* (CMU/SEI-96-TR-019) Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1996.
[Tilley 97]	Tilley, S. R. & Smith, D. B. "On Using the Web as Infrastructure for Reengineering," pages 170-173. *Proceedings of the 5th International Workshop on Program Comprehension (IWPC '97).* Dearborn, MI, May 28-30, 1997. Los Alamitos, CA: IEEE Computer Society Press, 1997.

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1. **AGENCY USE ONLY** (LEAVE BLANK)	2. **REPORT DATE** October 1997	3. **REPORT TYPE AND DATES COVERED** Final
4. **TITLE AND SUBTITLE** Discovering DISCOVER		5. **FUNDING NUMBERS** C — F19628-95-C-0003
6. **AUTHOR(S)** Scott R. Tilley		
7. **PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)** Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA 15213		8. **PERFORMING ORGANIZATION REPORT NUMBER** CMU/SEI-97-TR-012
9. **SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)** HQ ESC/AXS, 5 Eglin Street, Hanscom AFB, MA 01731-2116		10. **SPONSORING/MONITORING AGENCY REPORT NUMBER** ESC-TR-97-012
11. **SUPPLEMENTARY NOTES**		
12.A **DISTRIBUTION/AVAILABILITY STATEMENT** Unclassified/Unlimited, DTIC, NTIS		12.B **DISTRIBUTION CODE**

13. **ABSTRACT** (MAXIMUM 200 WORDS)

This report describes investigations into DISCOVER, a modern software development and maintenance environment. The study is guided by a framework for classifying program understanding tools that is based on a description of the canonical activities that are characteristic of the reverse engineering process. Implications of this work for advanced practitioners, researchers and tool developers, and the framework itself are discussed.

14. **SUBJECT TERMS** DISCOVER, framework, program understanding, reverse engineering			15. **NUMBER OF PAGES** 22
			16. **PRICE CODE**
17. **SECURITY CLASSIFICATION OF REPORT** UNCLASSIFIED	18. **SECURITY CLASSIFICATION OF THIS PAGE** UNCLASSIFIED	19. **SECURITY CLASSIFICATION OF ABSTRACT** UNCLASSIFIED	20. **LIMITATION OF ABSTRACT** UL